Politecnico di Milano

POLITECNICO
MILANO 1863

Financial Engineering

Group 19

Final project: Energy price forecasting

Authors:
Marco Giovanni Barbero
Lorenzo Biraghi
Roberto Giannini Braghè

June 9, 2024

Contents

1 Introduction to energy price forecasting

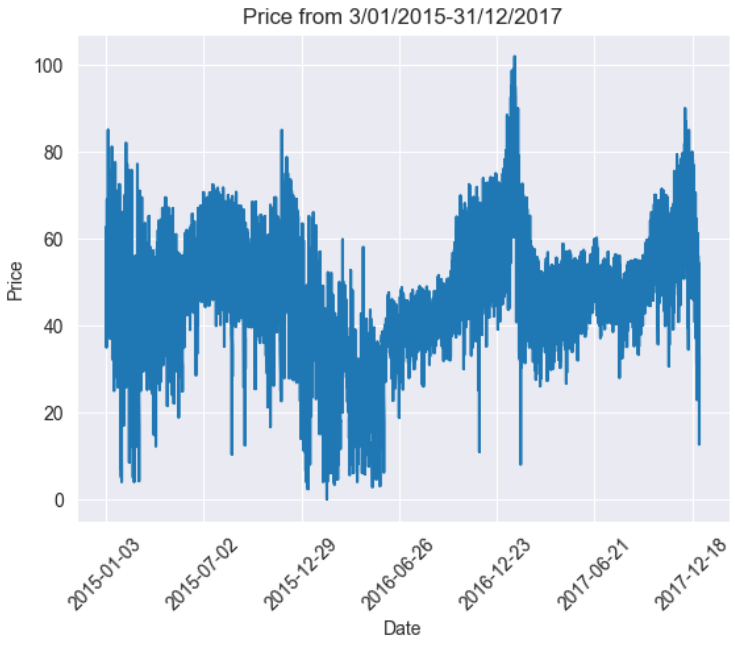
The liberalization of the electricity market from 1990 has created an increasing competitive energy market. There are different electricity market in Europe, but the most important one is the spot market and in particular the day-ahead auction.
It takes place at 11 AM everyday where all the aggregators bid the electricity price for all the 24 time slot of the next day.
Since the inception of this market, energy price forecasting has always been a relevant task for energy companies in the decision making system.
First the business of forecasting was tackled with point prediction using linear models and only quite recently with non-linear models, achieving by far better results.
However the recent integration of renewable energy production and smart grids resulted in greater uncertainty of future supply and future demands causing higher electricity price volatility and so the need a more comprehensive way of forecasting. Probabilistic forecasting is the solution of this problems since allows market participants the possibility of better managing their bidding during auctions and creating different possible strategy based on a probabilistic framework. Probabilistic forecasting is also the essence of risk management, crucial for energy trading firms in controlling their portfolios in the power sector. [3] [4]

2 Data exploration

The dataset is composed by 15 columns:

- $TARGEMPRICE$ the electricity energy price for each time slot.
- $FUTUEMload$ the electricity energy load forecast for the same time slot.
- $FUTUEMsolar$ and $FUTUEMwind$ the production energy forecast from solar and wind plants of the same time slot.
- Others feature are hours, dates and constants used in sin/cos to account for periodicity and seasonality.

We want to predict the day-ahead energy price, meaning that at day 1 we want to forecast the prices for the 24 time slot of day 2, and so for each day we have 24 different prices. Our dataset starts on 03/01/2015 and ends 31/12/2017, therefore we have almost three year for training and validate our models.
We begin the analysis looking at the prices on the whole period and their distribution.

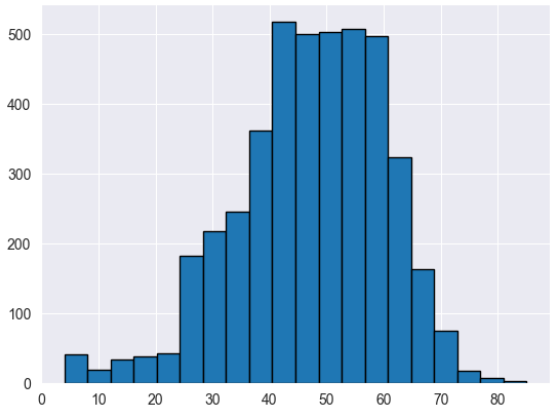


The price is highly volatile, with very huge price spikes. We can observe the seasonality of the price during the second part of winter 2014/15, the full winters of 2015/2016 and 2016/2017 and the first part of the 2017/2018. Also for the year 2015 and 2017 there are smaller peaks during summer explained by the demand for air conditioning.
The sample standard deviation normalized by the sample mean is equal to 30%, indeed a very high volatility.

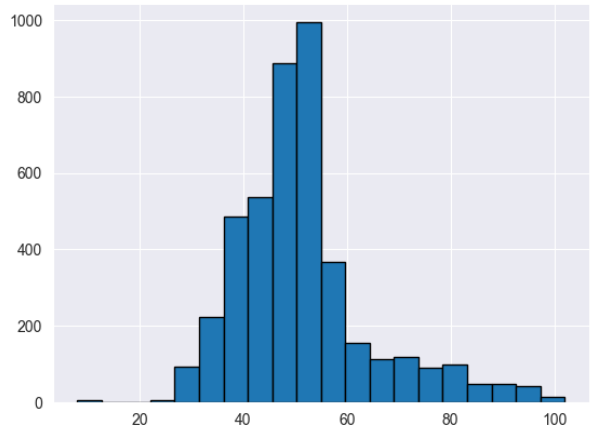
The distribution is a quite negative skewed bell curve centered at 47. We look over periods of six

months if this distribution is stable over time, if that's the case it will ease our forecasting task. We look at price plot and distribution on two different periods:

03/01/2015 - 30/06/2015

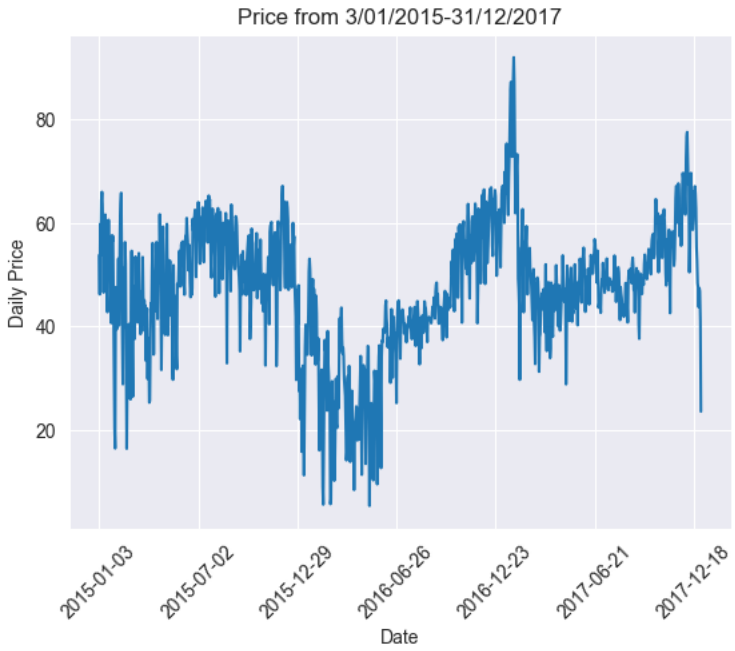


01/01/2017 - 30/06/2017

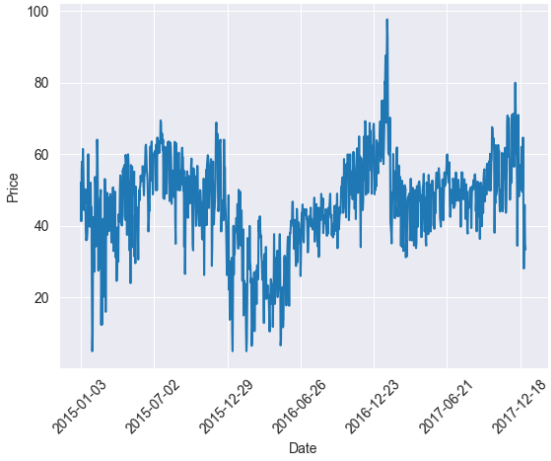


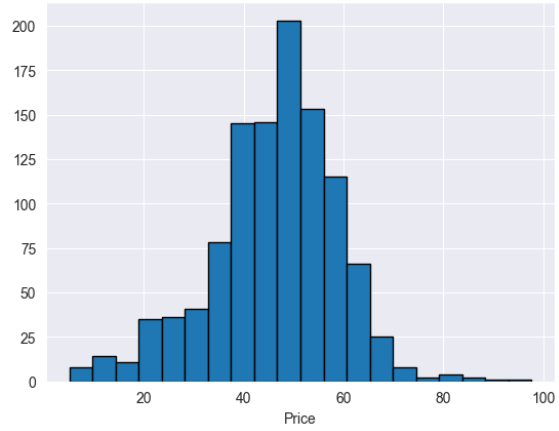
The two distributions are very different, although derived on the same months with different years, so we can derive that the distribution of the price process is not stationary in time and this poses a challenge from a forecasting point of view.

We want to have a less noisy plot of the prices on the whole period to get a sense of the trend, for this we compute the daily price, the average of the 24 slot prices of each day.

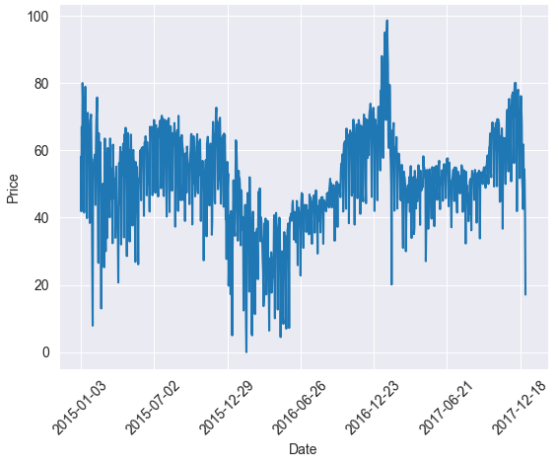


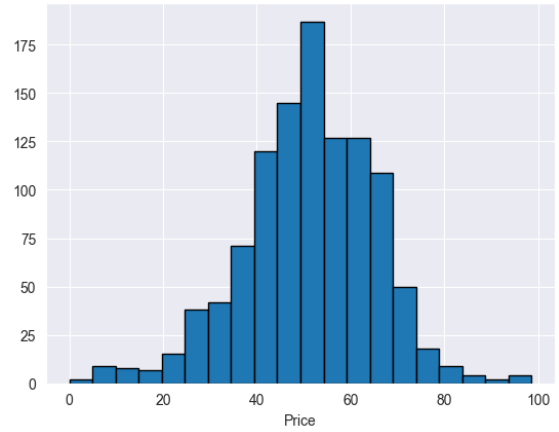
Now we want to explore if there are some regularities in each price slot, we achieve that by plotting the price and distribution of three different time slots far for each other.



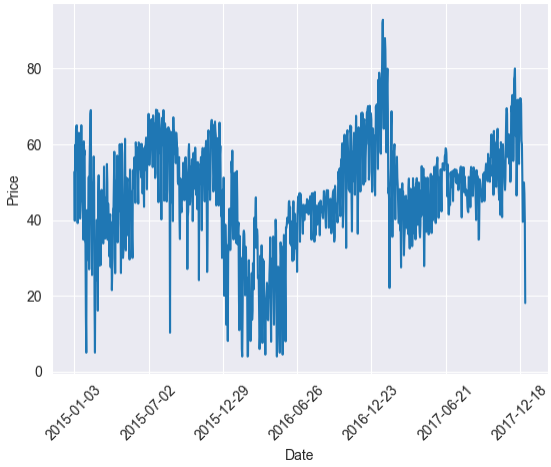


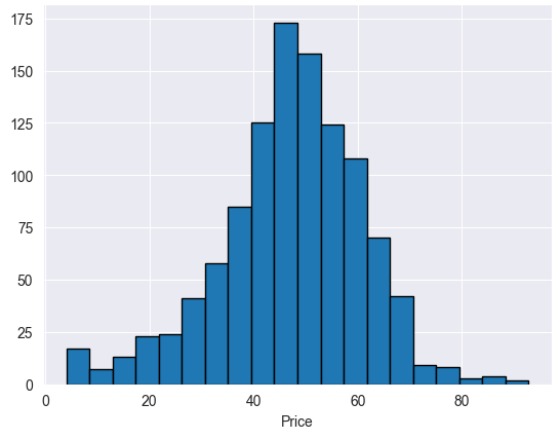
Time slot: 00:00-01:00





Time slot: 10:00-11:00

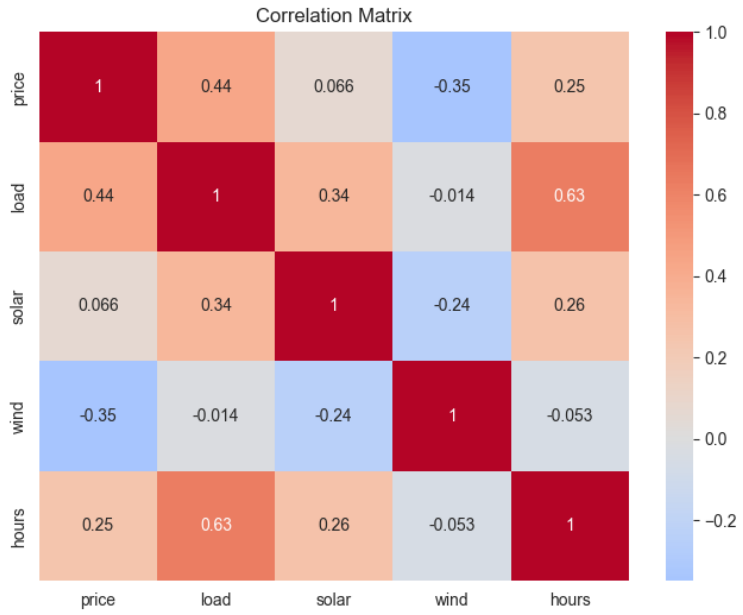




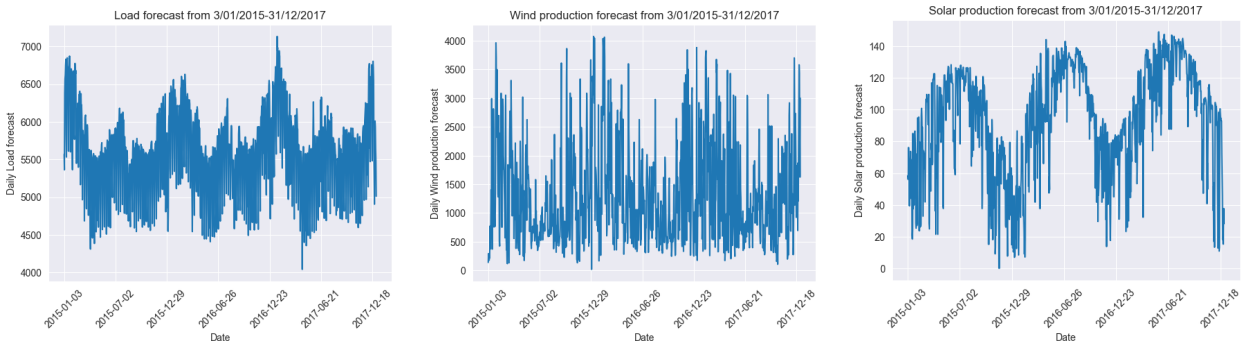
Time slot: 16:00-17:00

It's immediate to see that all the prices follow the same underlying trend, regardless of the hour: if we are in a period of high energy price all prices will be high with respect to their price range. We can also see that the distributions, overlooking a bit of variation in skewness, are quite similar, so it will be worthwhile basing a distributional prediction on each specific time slot.

We explore now the relations between the price and the given features, we accomplish this by computing a correlation matrix with price, solar, wind and hours. Looking at the correlation matrix we see that the feature most correlated with price is the load forecast, that is positive correlated since an higher demand for electricity will lead to higher prices in an offer/demand reasoning. Also the wind production forecast negatively impacts quite significantly the price, because it will bring an increased offer on the market. The hours also influence the price in fact even if all prices follow the same trend, as seen before, different time slots have different price ranges.



We look at the features over the whole dataset, with the caution of plotting the daily value of each feature to have some graphical interpretability.



The three plots are highly volatile, in particular we notice that the load trend is very similar to the price trend, the wind forecast is very noisy with high abrupt spikes and the solar forecast as we can imagine is a bit smoother and more cyclical.

3 General approach in model investigation

We look for the best model starting with ARX and distributional neural networks training with some standard conservative choices of hyperparameters(e.g learning rate $= 0.01/0.001$).
When results seem promising, as in the case of the ARX, we really try to go deeper in different ways: in hyperparameter tuning, experimenting various combination of optimizers other than Adam, such as AdamW, Adamax, SGD or strategies for adaptive learning rate during training like the keras

ReduceLROnPlateau.
We also tried some preprocessing of data, using the libraries included in keras, but no real benefit came from this since most of the methods we would use are already quite capable of dealing with inputs both processed and not. We also did not have categorical features, that are usually what requires the most attention in preprocessing.
The first model that came to mind when seeing the graph of hourly price was to use Distributional Networks for those predictions, but we quickly realized those yielded few results.
After this analysis we gradually increased model complexity like XGBoost, RNN & LSTM and at last Transformer. Speaking about this more complex model we only report some results since, even if we have done various modification all reported in the code, we didn't achieve any remarkable results and for the sake of brevity we didn't report all. All the results are reported on 2017, if not stated otherwise.

4 Forecasting models:

Forecasting models employ different degrees of probabilistic techniques and machine learning to predict the day-ahead price of electricity. In this case we are also interested in computing Confidence Intervals for the prediction, as a way to further guide our decision making.
In order to compare them to each other we could use:

- Root Mean Square Error (RMSE), a typical measure of accuracy of the exact prediction:

$$RMSE = \sqrt{\frac{\sum (y_{true} - y_{pred})^2}{N_{pred}}} \tag{1}$$

- Average Pinball Loss, also very common in the literature to evaluate the performance of quantile-based predictors, that penalizes quantiles larger than what is needed:

$$\mathcal{P}\left(\alpha, \hat{q}_{\alpha,n}, y_n\right) := \alpha \left[y_n - \hat{q}_{\alpha,n}\right] \mathbb{I}\left(y_n \geq \hat{q}_{\alpha,n}\right) + (1-\alpha)\left[\hat{q}_{\alpha,n} - y_n\right] \mathbb{I}\left(y_n < \hat{q}_{\alpha,n}\right) \tag{2}$$

$$\text{Pinball}(\alpha) := \frac{1}{N} \sum_{n=1}^{N} \mathcal{P}\left(\alpha, \hat{q}_{\alpha,t}, y_t\right) \tag{3}$$

- Delta Empirical Coverage, a method introduced for the back-testing of forecasting models that emphasizes reliability, the ability of the model to construct Confidence Intervals whose likelihood of a miss is exactly the Confidence Level:

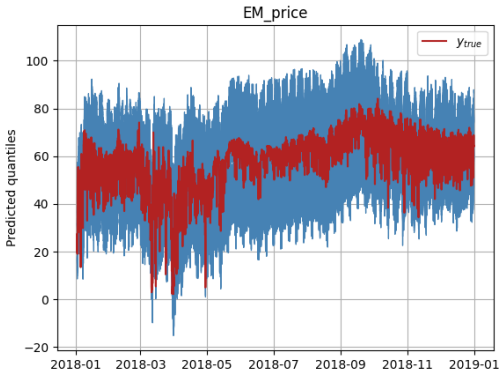
$$EC_\alpha := \bar{I}_t = \frac{1}{T} \sum_{t=1}^{T} \begin{cases} 1 & \text{if } y_t \in \left[\hat{L}_t, \hat{U}_t\right] \quad \text{"hit"} \\ 0 & \text{if } y_t \notin \left[\hat{L}_t, \hat{U}_t\right] \quad \text{"violation"} \end{cases} \tag{4}$$

$$\Delta C := \frac{1}{100 * (\alpha_{\text{Max}} - \alpha_{\text{min}})} \sum_{a=100*\alpha_{\text{min}}}^{100*\alpha_{\text{Max}}} |EC_a - a| \tag{5}$$

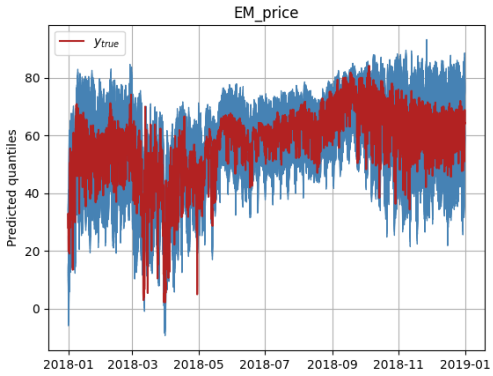
We do notice that in most of the cases the last two measures act against each other, Pinball seeking to reduce Confidence Interval size and Delta Coverage needing to find the more precise approximation of the distribution of the predicted quantity. Since the metric we are most interested in is Delta Coverage we will focus on that, and we will use RMSE to evaluate the accuracy of the point predictions.

4.1 Baseline model:

A baseline model is usually a simple and rapid predictor, accurate enough to make it a valid alternative to most other methods: if a new predictor is not able to consistently outperform the baseline it's not even worth considering.
First we will examine two models, ARX and DNN, how to generalize them to have quantile predictions and how to improve upon them.
The Autoregressive Exogenous model is a model that constructs a linear map of previous values of the input along with future parameters to make the next prediction, minimizing via ADAM optimization the loss function (MAE). We will then supplement the model with Conformal Prediction to obtain quantile predictions. Conformal Prediction is a Historical Simulation method that computes the quantiles of the price from a sample of past values, so that no assumptions on the distribution of data are necessary. This type of methods is preferred in finance due to the tendency of the markets to present extreme events more often than what classical bell curves predict, and due to how the strong link to reality of the values obtained.
Distributional Neural Networks on the other hand are a specific class of neural networks that fit the behaviour of their input to some distribution given a priori, in the simplest case let it be Gaussian. These will be our baseline, and we will show our main results on the electricity price prediction on the whole of 2018.
We immediately notice that Δ-Cov benefits larger Confidence Intervals, that allow for more leeway when making predictions, since it does not take into account in the metric the size of CI used. This is, again, opposite to the aim of Pinball loss, therefore using it to evaluate our model would prove counterproductive.
We also tried to give our predictor more than seven previous days for the prediction, then less, but both of those options decreased the accuracy of the model, and were abandoned soon.



(a) Base ARX - point prediction on 2018



Base N-DNN - quantile prediction on 2018

Model	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
point-ARX	5.062910, 1.232623	5.006487, 1.167935	5.035022, 1.294774
N-DNN	5.298960, 9.37975	5.036950, 10.31836	5.237802, 7.406139

Table 1: ARX with recalibration

4.2 ARX model:

As the first improvement we can apply, for predictions over an entire year, we can limit how often the method is recalibrated: we found that the method does not lose much accuracy by recalibrating every 28 days, and it makes the training significantly faster. This will be invaluable to test ulterior methods in the future, as it reduces the time required to obtain results by orders of magnitude, while also being a good compromise for accuracy. The results are for an ARX with quantile predictions run three times with the same parameters on 2017, with RMSE and Delta Coverage of each run and the total time to obtain results. As we can see the loss in accuracy is very slight, especially when compared to the time necessary for each run. We also notice that Delta Coverage can have some variance between the various runs, but the 28 day recalibration seems consistently the best.

Recalibration	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]	Approx Time
1/Day	4.429779, 0.423551	4.490516, 0.442415	4.455587, 0.408422	3 hours
1/Month	4.496192, 0.385337	4.543196, 0.463977	4.530044, 0.410705	15 minutes
1/Year	4.714849, 0.447488	4.740501, 0.452562	4.763974, 0.509639	2 minutes

Table 2: ARX with recalibration

Another improvement we can apply to ARX is the introduction of a regularizer, for example Lasso, a method to limit the size of non-fundamental weights. This method is not used to decrease overfit-

ting in the predictions, since it forces an additional constraint on the ability of the model to fit upon the training dataset. Some regularizers, mainly Lasso, increase decisively the interpretability of the method, and they also limit the likelihood of the predictor needlessly recalibrating unimportant weights for too little performance improvements. In the table below we will see the performance of the three regularizers, evaluated for predictions on the entire 2017, with a fixed learning rate of 10^{-4} for consistency and L1, L2 parameters found via Optuna optimization. Also for consistency's sake we will show the results of three separate runs with the same parameters. All the regularized ARX perform better than the base case with regards to RMSE, with Lasso as the best performing model, while on Δ-Cov the method with the best realization is L2, while the others perform like the best case of no regularization. Nevertheless we will choose L1 as our regulatizer in all the following modifications, since quantiles will be further refined whereas the accuracy of the central prediction is a more universally useful characteristic.

Regularizer	L1	L2	Test 1 [RMSE, Δ-Cov]	Test 2 [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
/	0	0	5.065633, 0.614916	5.040608, 0.465499	5.046659, 0.424911
Lasso	$9.430e-4$	0	4.539416, 0.462962	4.549234, 0.431760	4.525362, 0.465246
Ridge	0	$1.885e-2$	4.688255, 0.430999	4.692413, 0.362506	4.665576, 0.421106
Elastic Net	$6.150e-4$	$6.450e-3$	4.635360, 0.430492	4.593745, 0.451547	4.568944, 0.416539

Table 3: ARX with regularization

Now we turn to the main way we can compute quantile predictions from the ARX: conformal predictions. This method receives along with the ARX prediction a series of the previous prices, and uses those values to compute the empirical quantiles of the time series, and computes from that the Confidence Interval of the prediction. The size of this input is chosen a priori, and we will make a brief study on the optimal values like before. Four months of Conformal Predictions seem to be the best performing out of the three timeframes considered, but we bring to attention an interesting result that appears if we run the same test on the test set: the ranking of the three methods changes completely.

The reason why CP on 2018 performs so much better than it did on 2017 is due to how the training with conformal prediction happens: the sample that determines the size of the Confidence intervals is taken out of the training samples, so if we predict 2017 prices we are left with just 2015 for training, while 2016 is used for CP. This of course gives an extremely unreliable method. On 2018 predictions we have two years, we double the size of the training set, so we should expect a better accuracy, and it's the reason why we will consider it in the final iteration of this model. As for why the other methods have lost so much in Δ-Cov, it is probably due to 2018 having more spikes in prices during the first and last quarter of the year, such that confidence intervals computed on the previous timeframe is too strict and disallows the model from predicting the more frequent jumps of the new year. The CP computed on an entire year instead is able to retain memory of

the largest jumps, and it also gives insight into the behaviour of the price one year prior, very important especially for January that seems to always be highly stochastic.

CP size	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
122 days	4.539416, 0.462962	4.549234, 0.431760	4.525362, 0.465246
185 days	4.587669, 0.487942	4.623386, 0.411630	4.587745, 0.514652
365 days	4.714849, 0.447488	4.740501, 0.452562	4.763974, 0.509639

Table 4: ARX with conformal prediction on 2017

CP size	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
122 days	4.858213, 1.322678	4.813206, 1.375951	4.811867, 1.405124
185 days	4.819256, 0.850630	4.812562, 0.876068	4.765888, 0.854446
365 days	4.917326, 0.246067	4.904723, 0.200152	4.929607, 0.209030

Table 5: ARX with conformal prediction on 2018

Lastly we introduced a method to further reduce overfitting, widespread in the literature: a Dropout layer between the main ARX body and the prediction. This is a method that eliminates a given percentage of the input data, extracted randomly at each step during the training. This ensures that the model does not rely too much on a single or few features, and that it strives to extract every minutia from the behaviour of the model. Again, we let the results speak. All the results are computed with L1=9.430e-4, lr=1e-4, recalibration every 28 days on the entirety of 2017 and CP 185. It seems that introducing Dropout greater than 10% makes the model a lot less stochastic.

Dropout rate	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
0.0	4.526645, 0.372907	4.482213, 0.415778	4.510763, 0.463977
0.05	4.524667, 0.570614	4.522201, 0.464738	4.579176, 0.407661
0.10	4.597655, 0.386605	4.614190, 0.373921	4.593287, 0.406900
0.15	4.719717, 0.402587	4.762212, 0.389903	4.749327, 0.411973

Table 6: ARX with Dropout

Now we illustrate the **results on the Proto-test set** of the first model we chose, ARX point predictor with Conformal Prediction for the estimation of quantiles. The model uses a learning rate of 10^{-4}, L1 regularizer with weight $9.43 * 10^{-4}$, Conformal Prediction size of 365 and recalibration every 28 days. The model uses Dropout of 0.10 and on the time-series of 2018 has average metrics of:

$$\text{RMSE} = 4.863184 \qquad \Delta\text{-Cov} = 0.193064$$

Just for comparison, here are the Δ-Cov of the same model with and without Dropout:
Dropout 0.00: 0.370370, 0.233384, 0.246067
Dropout 0.15: 0.150177, 0.244799, 0.176306;
while a change in CP size to 122 gives:
CP 122: 2.390963, 2.607265, 2.712254.
This allows us to say with some certainty that the parameters we chose for ARX are some of the best, and definitely best among those we examined.

4.3 DNN model:

We come now to talk about DNN: in this case the prediction of quantiles is part of the method itself, in fact the point prediction is operated by a simple ANN, while the distributional part is used to fit to the data a certain distribution. So naturally the first way a DNN is improved is by the use of different distributions, for example Student's t and Johnson's SU distributions, whose tails can be adjusted to better represent the distribution of values observed. In our case we would likely prefer JSU, since the four input parameters allow for some more flexibility than Student's three, but both are an improvement over N-DNN. We also could resort to Quantile Regression, that foregoes the need to specify an expected distribution and uses data to simulate the quantiles that seem more appropriate, or use again Conformal Predictions on the point prediction, should the neural network be more accurate than ARX in the prediction. We will compare these models, like before, but first let's discuss some of the hyperparameters of choice.
First we tune the size of the network, the loss function and the and their activation functions. For each of the distributions we found these optimal parameters through Optuna tuning:

Method	Hidden layer size	Learning rate
N-DNN	128	10^{-3}
t-student-DNN	512	10^{-3}
JSU-DNN	512	10^{-3}
QR-DNN	64	10^{-3}
CP point-DNN	512	$5 * 10^{-4}$

Table 7: DNN tuned hyperparameters

The chosen loss function for each DNN based on known distributions was Log-likelihood, an estimator of the probability that the data belong to the probabilistic distribution examined. This allows the model to tune the parameters of each distribution to best fit the input and, if the distribution could accurately describe the data, would ensure the DNN to find the best approximation for it.
The loss used in point predictions is instead MAE. This should not surprise, since in point predictions the DNN is actually reverted to a simple neural network, and the estimation of quantiles is performed by Conformal Predictions like ARX before.
Lastly for Quantile Regression we use Pinball Loss, since it's the usual measure used to simulate

the values of the quantiles.
Now, excluding point-DNN that uses a different model altogeather to compute quantiles, the best basic distributional network is definitely Student's t. It is not surprising that it outclasses Normal, and also when compared to JSU the lower number of parameters may have played to its favor.

Method	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
N-DNN	5.905495, 4.167935	5.499191, 4.487569	5.281774, 4.245306
t-student-DNN	4.501092, 2.110603	4.502946, 2.814561	4.533377, 2.929984
JSU-DNN	4.727318, 4.086757	4.743827, 4.364535	4.739510, 3.732876
QR-DNN	5.134800, 3.517195	5.135886, 3.439051	5.185773, 4.429131
CP point-DNN	5.052091, 0.424911	5.064273, 0.486808	4.870556, 0.580669

Table 8: DNN methods results

Now we can introduce some improvements on these models as well, some similar to those applied to ARX, some unique to these models.
Unfortunately Normal-DNN has little chance of seeing improvements, since as we observed in the data analysis phase the prices are far from Gaussian, so we will skip to the next.
Student's t and JSU are somewhat similar both in their use as more flexible alternatives to Normal distribution and in their implementation. This also reflects in the methods we can use to modify them: beyond recalibrating less often, in particular we will choose again recalibration once every 28 days for each model, we can introduce either a Dropout layer or a Noise layer. Dropout as we discussed before disincentivizes overfitting by taking away random features from the predictor, Gaussian Noise achieves the same goal by modifying the values of inputs with a vector of noise Gaussian distributed. The results on 2017 show that the addition noise, while using both might result in some confusion as the data is excessively manipulated.

Method	Best parameter	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
Dropout	0.15	4.574054, 1.632420	4.499543, 2.234906	4.565277, 1.714865
Noise std	0.10	4.568905, 1.232876	4.471407, 2.762557	4.548145, 2.877980
Dropout+Noise	[0.10, 0.05]	4.620209, 3.009893	4.651580, 3.907914	4.612078, 3.682141

Table 9: Student's t distribution

Quantile Regression is a very powerful method, but the use of Pinball in the construction of the Confidence Intervals does work against the target metric of Delta Score. This could be solved by introducing a new loss function, that borrows from Pinball its main characteristics but does not give equal weight to near and far errors. We present Square-Pin loss, that we expect to run even worse on Δ-Cov since the errors of higher quantiles will hold even more weight in this loss, and

Rad-Pin loss, that applies the opposite transformation and should therefore be more lenient of farther predictions, which we hope will result in the larger quantiles Δ-Cov enjoys.

$$f_{sqr}\left(\hat{q}_{\alpha,n}, y_n\right) = \left(\hat{q}_{\alpha,n} - y_n\right)^2 \qquad f_{root}\left(\hat{q}_{\alpha,n}, y_n\right) = \sqrt{\left(\hat{q}_{\alpha,n} - y_n\right)} \tag{6}$$

$$\mathcal{P}\left(\alpha, \hat{q}_{\alpha,n}, y_n, f\right) := \alpha f\left(\hat{q}_{\alpha,n}, y_n\right) \mathbb{I}\left(y_n \geq \hat{q}_{\alpha,n}\right) + (1-\alpha) f\left(\hat{q}_{\alpha,n}, y_n\right) \mathbb{I}\left(y_n < \hat{q}_{\alpha,n}\right) \tag{7}$$

$$\text{SquarePin}(\alpha) := \frac{1}{N}\sum_{n=1}^{N} \mathcal{P}\left(\alpha, \hat{q}_{\alpha,t}, y_t, f_{sqr}\right) \qquad \text{RadPin}(\alpha) := \frac{1}{N}\sum_{n=1}^{N} \mathcal{P}\left(\alpha, \hat{q}_{\alpha,t}, y_t, f_{root}\right) \tag{8}$$

Comparing the results, it's clear that the loss functions influence the ability of the model to compute accurate quantiles. The biggest differesce is felt when switching to RadPin, that allows the model to keep larger Confidence Intervals with not much loss, and this is the way to improve our chances to not miss some unexpectedly large realizations of the price. SquarePin has the opposite effect, trying to adhere as close as possible to the expected quantiles and increasing the very sensitive Δ-Coverage.

Loss function	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
QR-DNN	5.134800, 3.517195	5.135886, 3.439051	5.185773, 4.429131
SquarePin	5.02625, 5.425537	5.114075, 5.784919	5.071136, 4.952452
RadPin	5.365186, 1.933490	5.873549, 2.032271	5.628792, 1.835351

Table 10: QR with custom loss

As for point prediction using DNN and quantile prediction with CP, much of the work to improve the second part has been done already. What we could still work on is trying to improve the point prediction as well. We tried changing the loss function, but with little results: the original choice of Mean Absolute Error was reasonable, we tried also MSE and SMAPE but the change was not large enough to justify one choice over the others. Moreover, we can see when we compare basic ARX and DNN point predictions supplemented by CP that the performance on Delta Coverage is actually very similar, even if the methods have little in common. This tells us that, for the size of the confidence intervals computed by the CP, it's not mandatory for the point prediction to be exact. We also observe that since Delta Coverage is very sensitive to predictions failing too often to capture the true price, a method like Conformal Prediction that keeps relatively wide Confidence Intervals during the year is preferable, even if it proves less informative about the actual future price. Here is the comparison between the two:

Model	Test 1: [RMSE, Δ-Cov]	Test 2: [RMSE, Δ-Cov]	Test 3 [RMSE, Δ-Cov]
ARX	4.526645, 0.372907	4.482213, 0.415778	4.510763, 0.463977
DNN	4.994267, 0.393201	5.052091, 0.424911	5.064273, 0.486808

Table 11: Point ARX vs Point DNN

RMSE of the two methods outlines that DNN is much more consistent in the error of its point prediction, but regardless of the loss function used it seems less capable than even a basic ARX to reach a better accuracy in the median prediction. It's no use comparing Delta coverage, since the two methods use the same Conformal Prediction, which is not even a stochastic method. Logically, and confirmed by our tests, there is no chance for DNN to perform the same optimizations of ARX and not maintain the same ranking.
Overall we can see that ARX, although simple, can rise to be one of the most accurate methods when manipulated enough. It is very stable when it comes to the error of the point prediction. It can, however, be a bit inconsistent when it comes to Delta Coverage, but until we show more interesting methods it earns its place among our candidate models. No basic DNN was able to rival its performance, but they will be a good starting point for the more complex methods we will present now.

5 XGBoost & LightGBM

We have seen that the linear model ARX with conformal prediction accomplish quite remarkable results, so knowing that simple models are able to capture the complexity needed for our task, we will try to improve our forecast with a gradual increase of the model complexity. We do this implementing XGBoost where the computational power lies in simple models, but it is leverage exploiting the ensemble approach.

Ensemble is a general technique that combines different simple machine learning models, called weak learners, in a way where each one does a simple task and then the final output is the aggregation of each single model output, this is usually accomplished using two standard methods: bagging or boosting.

XGBoost, also known as eXtreme Gradient Boosting, is a ML algorithm that uses ensemble learning. XGBoost, as the name suggest, uses boosting where every new weak learner corrects the errors of the existing learners.
In particular XGBoost implements a random forest on the residual obtained by the first weak leaner, the output of the random forest corrects the forecast of the first weak learner and the result is then passed at the second weak learner that will have another random forest implemented on the new residual. This scheme goes on in an iterative way and is able to scale errors very quickly. Since the computational core of the model is simple the time needed for training is usually much

less than complex model like Recurrent Neural Networks or Transformer.

LightGBM works very similar to XGBoost, but is significantly lighter parameter-wise, in fact it is able to train much quicker resulting an accuracy very similar to XGBoost.
This is accomplish by leaf-wise growth of the random trees, whereas XGBoost focus on level-wise growth.

Model	CP Δ-Cov	CP exp. weights Δ-Cov	RMSE
XGBoost	2.21458	1.10125	4.78217
LightGBM	1.30257	0.79874	4.20163

Table 12: XGBoost and LightGBM point with conformal prediction

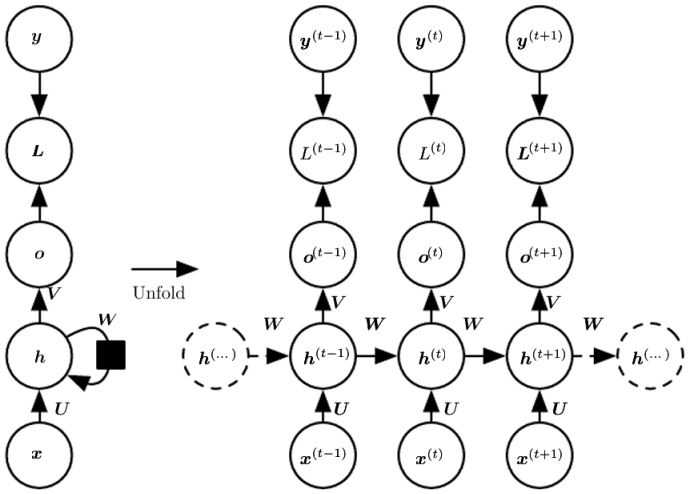
The results are worse than baseline models, so we did not investigate much further. This class of models is very used in practice because they are flexible and not very complex, but usually are trained with very long time series and often with many different realization of them, so since our data sample is quite small it is reasonable that do not works as well as in the you could expect by reading the literature. [8]

6 Recurrent Neural Network

Recurrent neural networks are networks that aim to processes a sequence of input values, such as a time series. Going from a classical multi-layer network to a recurrent network one key ingredient is needed: a new way of sharing parameters across different parts of the model. In recurrent neural networks each member of the output is a function of the previous members of the output and each output is produced by using the same update rule corrected by the exogenous input of the current time step. We can write that each hidden unit of the network is represented by the following equation:

$$h^t = f(h^{t-1}, x^t; \theta) \tag{9}$$

where h^t represent the state layer at time t, x^t is the exogenous input at time t and θ is a general parameter that describe the structure of h^t. We can represent this recursive processes in a unfolding graph and this allowing to view the whole evolution:



Here previous prices are x, predicted are o and y are the true prices. Our implemented structure is a bit different from this since all feature, here not present, and previous prices enter altogether in a dense layer before going into the stage of evaluation L, where the loss function is minimized.

6.1 Long Short-Term Memory

The main problem of recursive neural networks when learning over a long term period is due to the gradient, computed during the minimization of the loss function, that looping over layers many times will either explode or vanishes.

Long Short-Term Memory (LSTM) and GRU (gated recurrent unit) models tries to solve this issue by operating at different time horizon, in a way that some parts deal with fine-grained scales and handle small details while others parts operate at coarse time scales and transfer information from distant past to the present more efficiently.

A possible way to make the network remember past information is to have units with linear self-connections and a weight near one on these connections. For example when we accumulate a running average $\mu(t)$ of some value $v(t)$ by applying the update $\mu(t) \leftarrow \alpha\mu(t-1) + (1-\alpha)v(t)$, the α parameter is an example of a linear self-connection from $\mu(t-1)$ to $\mu(t)$.

When α is near one, the running average remembers information about the past for a long time, and when α is near zero, information about the past is rapidly discarded. Hidden units with linear self-connections can behave similarly to such running averages and are called leaky units. The power of LSTM lies in the ability to understand when the past information is no longer useful and forget it by setting α at zero, instead when information is still valuable so has to be remembered α needs to be set at one.

We tried different possible configuration both for the RNN and LSTM, here we report using the

best configuration some significant results using point prediction, recalibrated every 365 days, with conformal prediction both in the standard setting and with exponential weights.

Model	CP Δ-Cov	CP exp. weights Δ-Cov	RMSE
RNN	0.83771	0.73535	5.78217
LSTM	1.00963	0.88452	6.20163

Table 13: RNN and LSTM point with conformal prediction

We see that the Conformal Prediction standard and with exponential weights yields very similar results for RNN and quite different for the LSTM. The RNN Δ-Coverage score is good, but the RNN is a quite heavy configuration and also we observed that is not very stable, probably would work much better with a bigger loads of training data.

We also write the results, with 365 days recalibration, with quantile regression, normal distribution and t-student distribution

Model	Q.R. [Δ-Cov, RMSE]	Normal [Δ-Cov, RMSE]	t-Student [Δ-Cov, RMSE]
RNN	9.28741; 6.88725	0.96590; 6.11724	2.08777; 6.29720
LSTM	3.05377; 6.23169	3.98148; 5.54562	1.73008; 5,03148

Table 14: RNN and LSTM with QR and Distributional Normal and t-Student

We did not obtain any interesting value of Δ-Coverage, also RNN and LSTM differs on Quantile regression and Normal, but are quite similar for t-student distribution.
The best results we obtained is LSTM with t-Student distribution confirming heavy-tails distribution as the best way to model volatile markets with a non-negligible probability of very extreme values.
We recalibrated every year because increasing the number of recalibration yielded worse Δ-Coverage results and also significantly increased the computational time.

6.2 Failed attempts

By reading the literature on time series point forecasting, since conformal prediction seemed promising for our task, we found some models that were promising although significantly more complex then the best models we had. So we tried implementing them, knowing that they might perform poorly. For example we tried $BiTCN$[5], two Bidirectional Temporal Convolutional Networks, that performed poorly as expected. We also tried simplifying the model by using only one Temporal convolutional layer but the results were still way worse then the other benchmark models when applied as point prediction, distributional network and quantile regressor. So we quickly discarded this models.

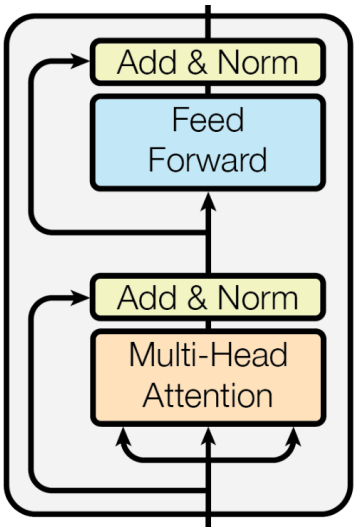
7 Additional considerations

In the beginning we considered also Pinball loss and Winkler score as metric to evaluate our models, but quickly realized that they were mostly uncorrelated to the Δ-Cov of the model, so we stopped reporting them for brevity. Also as previously noted Δ-Cov tends to have a relatively high variance even when comparing the same models and same hyperparameters on different runs. So we started taking into consideration also the changes in this metric between different runs, looking for models that were as robust as possible to this phenomenon. Also in the beginning we were evaluating the performances of various models monthly, but later we realized that sometimes models performed poorly on a particular month but then performed way better on the whole year. This happens since in a month were energy prices are noisy and changing rapidly a given model might make prediction intervals that are too narrow when compared to the actual price movements, but if in the following months

8 Gradually increasing model complexity

Given the limited amount of data, when compared to typical deep neural network applications, we started from extremely small models and slowly but gradually tried increasing complexity. Then as complexity of the models increases the most natural candidates were recurrent neural networks, given the nature of the problem.

We also explored a parallel set of models, in particular transformer based models, more precisely models using the encoder part of transformers [6]. We focused solely on the encoder part since, the decoder part would have resulted in models way to complex. So initially we built a small customized encoder layer composed of layer normalization, multi-head attention layer, and finally the feed forward part composed by a further layer normalizaton followed by convolutional layers. The main idea behind this choice was to build a deeper network but with very few additional trainable parameters compared to ARX, since the initial findings suggested that smaller models were more suitable for our problem, while the training time increases greatly.



Firstly we tried using this architecture on the dataset composed by the last seven days of historical time series data and the day ahead forecasts for load, solar and wind production. We also tried adding in the model the same forecast data but also for the 7 days of historical time series inputs, then rearranging the inputs in two blocks, a tensor of shape $(Batch\ size, 168, 4)$ with all the historical prices and forecasts, and a tensor $(Batch\ size, 192, 3)$ with the past and day ahead forecasts. All this additional features didn't give performance improvements on any of the considered metric and this behaviuor was consistent for all experimented models(even ARX for example) so with stopped considering the additional inputs.

8.1 Decomposition layer

We also tried applying a layer to decompose the time series data into trend and seasonal components.We did so by padding the extremes of our input time series and then applying global average pooling to get the trend, then subtracting the trend from the time series to get the seasonal component, as it was also done in the autoformer architecture[7]. We then fed the 2 split inputs form the time series into the nonlinear layers, and we observed that on most models this initial step gave a relatively small improvement in performances but has the additional benefit of giving more stable prediction on different runs, effectively reducing the stochastic component in the training and therefore performances of our models.

We will refer to the model composed by the trend-seasonal decomposition layer, followed by our custom encoder layer applied to the trend, seasonal and exogenous components that are then concatenated in the last output dense layer as $DECTRANSF$. We report this specific model results since they were the most promising among all the models using the encoder layer, and after trying various configurations we obtained the following on 2017:

Model	point: [RMSE, Δ-Cov]	QR: [RMSE, Δ-Cov]
DECTRANSF	4.457498, 0.25362	4.910220, 7.258803

Model	Normal: [RMSE, Δ-Cov]	t-student: [RMSE, Δ-Cov]	JSU: [RMSE, Δ-Cov]
DECTRANSF	4.968523, 5.679922	4.623954, 2.256113	4.582085, 0.694317

Table 15: DECTRANSF on 2017

Then since the model was promising we also evaluated it on 2018(we skipped qr and Normal since they were worse):

Model	point: [RMSE, Δ-Cov]	t-student: [RMSE, Δ-Cov]	JSU: [RMSE, Δ-Cov]
DECTRANSF	5.48255, 0.220991	4.768523, 0.451997	4.848060, 2.463825

Table 16: DECTRANSF on 2018

Model	point(Test 1): [RMSE, Δ-Cov]	point(Test 2): [RMSE, Δ-Cov]	point(Test 3): [RMSE, Δ-Cov]
DECTRANSF	5.27441; 0.20785	5.11247; 0.29821	4.98185; 0.16987

Table 17: RESIDUAL DENSE on 2018

As we did for RNN's we used this models for probabilistic forecasting, quantile regression and as distributional neural network. We note that since the training time of this encoder model is significantly greater we mainly evaluated the performances for the different parameters and hyperparameters recalibrating only once a year, then tried with more frequent calibrations for the most promising models. We additionaly note that the point forecast with delta coverage was done with a calibration bag of size 184. Generally we saw that increasing the calibration bag of the other models improved the delta coverage, as long as removing more data from the training set dosen't negatively affect the model training. For the DECTRANSF this indeed happend, as both on 2017 and on 2018 using a recalibration bag of 365 worsen dramatically the model as the delta coverage on 2018 oscillates around 1, and think that given a larger dataset this model would benefit from it, possibly making the choice of a recalibration bag of 365 viable also for this model. Furthermore we note that as already seen in RNN's also in this case recalibating the model too often causes an increase in delta coverage while the RMSE remains consistent. Since this critical issue with the delta coverage is common to many models that perform probabilistic forecasting thanks to the Conformal prediction, we suspect that this is an intrinsic problem of conformal prediction when evaluated with the delta coverage.

8.2 Nonexchangeable conformal prediction

After having observed that for predictions spanning for a whole year the conformal prediction methods were generally yielding the best results we also decided to try to tackle the critical points of applying conformal prediction to time series forecasting. The main issue lies in the exchangeability assumption needed for cp, hence we tried implementing a variant taking inspiration from the paper [1], where the main idea is based on the intuition that for time series the error at the current prediction step will follow a distribution which is more similar to the closer in time past samples. Hence we assign weights, decreasing as the time difference increases, then normazlize them to have them sum to 1.

For conformal prediction we estimate the distribution for the confidence interval at the new time step as $\widehat{C}_n(X_{n+1}) = \widehat{\mu}(X_{n+1}) \pm (\text{ the } \lceil (1-\alpha)(n+1) \rceil\text{-th smallest of } R_1, \ldots, R_n)$ Equivalently, we can write

$$\widehat{C}_n(X_{n+1}) = \widehat{\mu}(X_{n+1}) \pm \mathrm{Q}_{1-\alpha}\left(\sum_{i=1}^{n} \frac{1}{n+1} \cdot \delta_{R_i}\right),$$

Instead we now we estiamte the distribution by a weighted average of the delta functions:

$$\widehat{C}_n(X_{n+1}) = \widehat{\mu}(X_{n+1}) \pm \mathrm{Q}_{1-\alpha}\left(\sum_{i=1}^{n} \tilde{w}_i \cdot \delta_{R_i}\right),$$

Where δ_{R_i} is the Dirac-delta centered in $R_i, R_i = |\widehat{Y}_i - Y_i|$ and $\widehat{Y}_i$ is the forecast at time i.

Then the main challenge would be the choice of the weights, but following the intuition presented above we define them as $w_i = 0.99^{n+1-i}$ and $\tilde{w}_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$. We note that this change doesn't guarantee to solve the exchangeability issue.

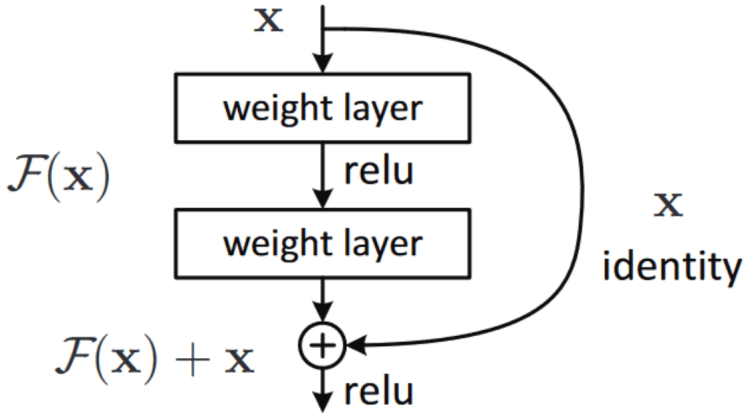
8.3 Results of the new conformal prediction

We first start by noticing that considering $DECTRANSF$ for point prediction on the 2017, the Δ-Cov was on average between 0.20 and 0.35 when using the original conformal prediction, while when using the weighted conformal was between 0.10 and 0.21, and on average 0.1632. So this seemed to improved greatly our performance, and all other point models(ARX, DNN, RNN's, etc.) confirmed this trend when considering the whole 2017. Later seeing the good performance of $DECTRANSF$ we decided to evaluate it's performances on 2018, obtaining 0.22(as reported above) with the original CP while 0.35 with the weighted one. This tendency was once again confirmed by other models so we deduced that this change is really dependant on the price trends of each year, and therefore unreliable. We also tried varying the exponential base of the weights that in the paper was set to 0.99, but when we decreased it to 0.985 it just amplified the differences in performance to the extremes, both positively and negatively, while when increasing it to 0.995 it either reduced

the differences, even though they still remained, or made no significant impact when compared to the original CP.

9 Residual Networks

Seeing the good performances of the networks using the encoder layer we wondered if the key factor in that model was the multi head attention layer, or the skip connection implemented by additions of the inputs to the outputs of the hidden layers in the encoder block. So, also considering the extremely slow training time for the $DECTRANSF$, we tried building other skip residual blocks, following the structures in the encoder or in the famous ResNet [2]. So we implemented various residual blocks, with two(or sometimes one) hidden layers on which we sum directly the inputs and add layer normalization.



We built the residual blocks with dense and convolutional hidden layers, and in the end also tried RNN layers, but the one giving the best results was the block with the dense layers. In the end the model that gave all its inputs to the dense residual block, regularized with dropout, that then fed its outputs to the last output layers had the following results:

Model	point: [RMSE, Δ-Cov]	QR: [RMSE, Δ-Cov]
RESIDUAL	4.769901, 0.393022	5.346499 , 16.461187

Model	Normal: [RMSE, Δ-Cov]	t-student: [RMSE, Δ-Cov]	JSU: [RMSE, Δ-Cov]
RESIDUAL	4.570011, 1.146169	4.644303, 3.264789	6.736520,2.561735

Table 18: RESIDUAL DENSE on 2017

So evaluating on 2018 (recalibrating every 28 days) only the point model, since was giving by far

the best results we got:

Model	point(Test 1): [RMSE, Δ-Cov]	point(Test 2): [RMSE, Δ-Cov]	point(Test 3): [RMSE, Δ-Cov]
RESIDUAL	4.924376, 0.25700	5.021356, 0.356418	4.954418, 0.262370

Table 19: RESIDUAL DENSE on 2018

So this new model could never outperform $DECTARNSF$ (or even ARX), but managed to come quite close to it, but even though is significantly larger in terms of parameters, is exponentially faster in terms of training time.

9.1 Ensembling

As already noted before, it came to our attention that many models gave very different results on different runs, and despite various attempts with regularization this problem persisted. So one natural solution was the implementation of ensemble methods, in the sense of training the same models numerous times and then average its predictions in order to have more stable results.
We also tried ensembling different model, our idea was to combine two models very different in nature, so our choices naturally went to the best performing simpler model, the ARX, and one of our best performing more complex model but that had a training time that allowed for numerous training instances, so the last Residual Dense model. We immediately saw that ensembling two different methods yielded worse results then any of the two methods alone. This also may be due to the particular choice of models combined, but nonetheless we decided to stop trying to ensemble different models. But we observed that in many cases, in particular with ARX, ensembling different training instances of the same model proved efficient to lower the variability of the performance measured with Δ-Cov.

10 Final Model

Our $DECTRANSF$ model managed to match the performance of the best baseline modell, which was the ARX with dropout 0.1.

We explored many state of the art models: from Recurrent Neural Networks to Gradient boost, to Residual Networks and a DECTRANSF. We compared their performance on 2017, 2018 and on single month of them, like January, May and November. We found most of those models to report huge differences in Δ-Coverage month-wise and with very unstable behaviour, whereas when we considered longer period, like one year, the metric become more stable. We expected this because a back-testing method like Delta Empirical Coverage benefits from larger sample size, and that on smaller time-frames, like a month, results can be quite erratic.

The general poor performance of all this very sophisticated models we think is due to the size of the dataset used, in fact a couple of years of data is a very small amount when compared to the dataset these state-of-the-art models are trained on, so the complexity that should be their strength becomes more of an hindrance.

The one model that was among all able to distinguish itself, being accurate and quite stable, was the DECTRANSF, in fact due to its resilient structure was able to adapt to the various tests.

In the end the model we choose is the one that suffers least the lack of data, managing to be quite stable and accurate like the DECTRANSF, is an ARX ensamble.

This model, accurately tuned, polished with a Dropout layer and a Conformal Predictor spanning the entire previous year, was able to deliver stable performance, and leveraging its simplicity allowed for more frequent recalibrations, pushing the ARX to improve by a large margin the accuracy of its predictions.

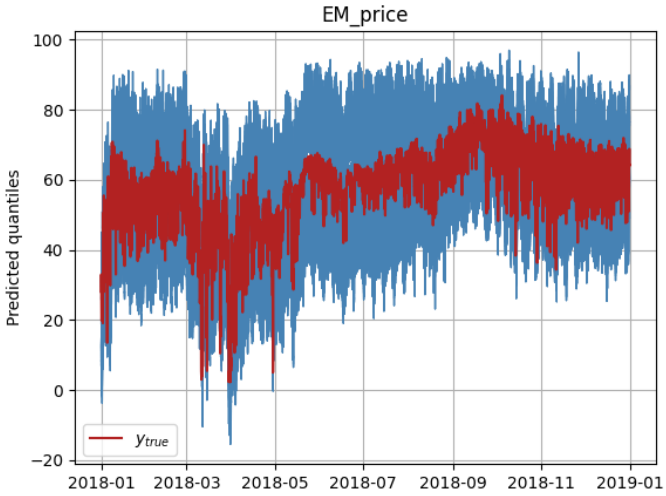
In the end the performance on 2018 of the ensemble-ARX and of the DECTRANSF were very similar, with DECTRANSF able to achieve lower Δ-Coverage scores in many of its realizations but losing in stability compared to the ARX ensamble.

Ultimately, our choice was driven by the need for consistency and more importantly of parsimony: among two methods whose performances are comparable, the costs must be taken into consideration, and the cost of DECTRANSF both in predictability and sheer time were deemed too high. So we couldn't justify in any way the choice of this more complex and advanced model even if extremely similar performance wise.
This is the final performance of the model, its description and its parameters:

$$\text{RMSE} = 4.863184 \qquad \Delta\text{-Cov} = 0.193064$$

Method	Ensemble-ARX
Ensemble size	4
Prediction horizon	24h
Learning Rate	1e-4
Regularizer	L1
L1 parameter	1e-3
Recalibration	28
CP size	365
Dropout	0.10

Table 20: Ensemble-ARX parameters



References

[1] Rina Foygel Barber et al. "Conformal prediction beyond exchangeability". In: *arXiv preprint arXiv:2106.00170* (2021).

[2] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[3] Grzegorz Marcjasz et al. "Distributional Neural Networks for Electricity Price Forecasting". In: *Preprint submitted to Elsevier* (2022).

[4] Jakub Nowotarski and Rafał Weron. "Recent advances in electricity price forecasting: A review of probabilistic forecasting". In: *Renewable and Sustainable Energy Reviews* (2018).

[5] Olivier Sprangers, Sebastian Schelter, and Maarten de Rijke. "Parameter Efficient Deep Probabilistic Forecasting". In: *International Journal of Forecasting* (2021).

[6] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[7] Haixu Wu et al. "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*. IJCAI, 2021.

[8] Wanlei Xue Yihang Zhao Huiru Zhao Xin Zhao Qiushuang Li and Sen Guo. "Research on Ultra-Short-Term Load Forecasting Based on Real-Time Electricity Price and Window-Based XGBoost Model". In: *Energies* (2022).